CLAIMS

1. A method for developing a software system, comprising the steps of:

providing an object model expressed in an object modeling computer language, the object model representing a software system and comprising components realizing classes;

providing a set of one or more metaprograms reflecting a computer system architecture; and

a meta-machine binding the components to the metaprograms to generate the software system for a computer system having said architecture.

2. The method claimed in claim 1, wherein the object modeling computer language is the Unified Modeling Language (UML).

3. The method claimed in claim 1, wherein:

the step of providing an object model is performed in part by a business analyst creating the object model; and

the step of providing a set of one or more metaprograms is performed in part by a technologist coding the metaprograms.

4. The method claimed in claim 1, wherein:

the step of providing an object model comprises a user using a graphical user interface to list a project in a first window, the project representing the object model; and

the step of providing one or more metaprograms comprises a user using a graphical user interface to list one or more metaprojects in a second window, each metaproject including a list of representations of the metaprograms.

5. The method claimed in claim 4, wherein the object modeling computer language includes an extension mechanism for the specification of user-defined properties and the assignment of these properties and their values to elements of the object model.

6. The method claimed in claim 5, wherein:

the object modeling computer language is the Unified Modeling Language (UML); and

the user-defined extension mechanism is a Stereootype.

7. The method claimed in claim 5, wherein:

the object modeling computer language is the Unified Modeling Language (UML); and

the user-defined extension mechanism is Tagged Values.

8. The method claimed in claim 5, wherein the step of a meta-machine binding the components to the metaprograms comprises the steps of:

searching the list of metaprojects for a metaproject having a name matching a name of an implementation target, wherein the implementation target is defined by the user-defined extension mechanism associated with the components; and

storing an indication of an association between the metaproject having the matching name with metaprograms of which representations are listed in the metaproject having the matching name.

9. The method claimed in claim 8, wherein:

the object modeling computer language is the Unified Modeling Language (UML);

the user-defined extension mechanism is a Stereootype; and

the Stereotype indicates the implementation target.

10. The method claimed in claim 8, wherein:

the object modeling computer language is the Unified Modeling Language (UML);

the user-defined extension mechanism is Tagged Values; and

the Tagged Values indicate the implementation target.

11. The method claimed in claim 1, further comprising the step of a user using a graphical user interface to invoke a metaprogram editor.

12. The method claimed in claim 1, wherein each metaprogram in said set of metaprograms includes code and metacode, and the metacode generates a portion of the source code of the software system by outputting the code.

13. The method claimed in claim 12, further comprising the step of a user using a graphical user interface to invoke a metaprogram editor.

14. The method claimed in claim 13, further comprising the step of a user activating a toggling function of the metaprogram editor to toggle a window between highlighting the code and highlighting the metacode.

15. The method claimed in claim 1, wherein the set of metaprograms includes a model metaprogram that modifies the object model.

16. The method claimed in claim 1, wherein the set of metaprograms includes a component metaprogram invoked once for each component and uses the classes realized by the component to produce a portion of the software system.

17. The method claimed in claim 1, wherein the set of metaprograms includes a class metaprogram that is invoked once for each class realized in each component and that produces a portion of the software system.

18. A meta-development environment computer program product for developing a software system, the computer program product comprising a computer-readable medium carrying thereon:

a meta-machine responsive to an object model and a set of one or more metaprograms input to the meta-machine under control of a user, the object model expressed in an object modeling computer language, the object model representing a software system and comprising components realizing object classes, the set of metaprograms reflecting a computer system architecture, the meta-machine under control of the user binding the components to the metaprograms to generate a resultant software system executable on a computer system having the architecture; and

a user interface for providing user control of metaprogram input, object model input, meta-machine operation, and resultant software system output.

19. The computer program product claimed in claim 18, wherein the object modeling computer language is the Unified Modeling Language (UML).

20. The computer program product claimed in claim 18, wherein the user interface comprises a graphical user interface listing a project in a first window, the project representing the object model, and listing one or more metaprojects in a second window, each metaproject including a list of representations of the metaprograms.

21. The computer program product claimed in claim 20, wherein the object

modeling computer language includes an extension mechanism for the specification of user-defined properties and the assignment of these properties and their values to elements of the object model.

22. The computer program product claimed in claim 21, wherein:

the object modeling computer language is the Unified Modeling Language (UML); and

the user-defined extension mechanism is a Stereootype.

23. The computer program product claimed in claim 21, wherein:

the object modeling computer language is the Unified Modeling Language (UML); and

the user-defined extension mechanism is Tagged Values.

24. The computer program product claimed in claim 21, wherein the meta-machine binds the components to the metaprograms by searching the list of metaprojects for a metaproject having a name matching a name of an implementation target and storing an indication of an association between the metaproject having the matching name with metaprograms of which representations are listed in the metaproject having the matching name, wherein the implementation target is defined by the user-defined extension mechanism associated with the components.

25. The computer program product claimed in claim 24, wherein:

the object modeling computer language is the Unified Modeling Language (UML);

the user-defined extension mechanism is a Stereotype; and

the Stereotype indicates the implementation target.

26. The computer program product claimed in claim 24, wherein:

the object modeling computer language is the Unified Modeling Language (UML);

the user-defined extension mechanism is Tagged Values; and

the Tagged Values indicate the implementation target.

27. The computer program product claimed in claim 18, wherein the user interface includes a metaprogram editor.

28. The computer program product claimed in claim 18, wherein each metaprogram in said set of metaprograms includes code and metacode, and the metacode generates a portion of the source code of the software system by outputting the code.

29. The computer program product claimed in claim 28, wherein the user interface includes a metaprogram editor.

30. The computer program product claimed in claim 29, wherein the metaprogram editor includes a toggling means for toggling a window between highlighting the code and highlighting the metacode.

31. The computer program product claimed in claim 18, wherein the set of metaprograms includes a model metaprogram that modifies the object model.

32. The computer program product claimed in claim 18, wherein the set of metaprograms includes a component metaprogram invoked once for each component and uses the classes realized by the component to produce a portion of the software system.

33. The computer program product claimed in claim 18, wherein the set of metaprograms includes a class metaprogram that is invoked once for each class realized in each component and that produces a portion of the software system.

34. A method for developing software systems for a plurality of computer system architectures, comprising the steps of:

providing a meta-development environment (MDE) to a first party having access to a first computer system with a first architecture;

providing the MDE to a second party having access to a second computer system with a second architecture;

providing the first party's MDE with an object model expressed in an object modeling computer language, the object model representing a software system and comprising components realizing classes;

providing the second party's MDE with the object model;

providing the first party's MDE with a first set of one or more metaprograms reflecting the first architecture, wherein binding in the MDE of the components of the object model and the first set of metaprograms defines first software system code executable on a computer system having the first architecture but not executable on a computer system having the second architecture, the first software system code effecting a user application when executed on the first computer system; and

providing the second party's MDE with a second set of one or more metaprograms reflecting the second architecture, wherein binding in the MDE of the components of the object model and the second set of metaprograms defines second software system code executable on a computer system having the second architecture but not executable on a computer system having the first architecture, the second software system code effecting the same user application when executed on the second computer system as the first software system code effects when executed on

the first computer system.

35. The method claimed in claim 34, wherein:

the MDE includes a metaprogram editor and a meta-machine;

the step of providing the first party's MDE with a first set of one or more metaprograms includes the step of the first party using the metaprogram editor of the first party's MDE to write metaprogram code; and

the step of providing the second party's MDE with a second set of one or more metaprograms includes the step of the second party using the metaprogram editor of the second party's MDE to write metaprogram code.

36. The method claimed in claim 35, wherein the object modeling computer language is the Unified Modeling Language (UML).

37. The method claimed in claim 35, wherein:

the step of providing the first party's MDE with an object model comprises the first party purchasing the object model from a vendor and loading the object model into the first party's MDE; and

the step of providing the second party's MDE with an object model comprises the second party purchasing the same object model from a vendor and loading the object model into the second party's MDE.

38. A method for developing software systems for a plurality of computer system architectures, comprising the steps of:

providing a meta-development environment (MDE) to a first party having access to a first computer system having an architecture;

providing the MDE to a second party having access to a second computer system having the architecture;

the first party writing a first object model expressed in an object modeling computer language and loading the first object model into the first party's MDE, the object model representing a first software system and comprising components realizing classes;

the second party writing a second object model expressed in an object modeling computer language and loading the second object model into the second party's MDE, the object model representing a second software system and comprising components realizing classes;

the first party purchasing from a vendor a set of one or more metaprograms reflecting the architecture and loading the set of metaprograms into the first party's MDE, wherein binding in the MDE of the components of the first object model and the set of metaprograms defines first software system code executable on a computer system having the architecture but not executable on a computer system having another architecture, the first software system code effecting a first user application when executed on the first computer system; and

the second party purchasing from a vendor a set of one or more metaprograms reflecting the architecture and loading the set of metaprograms into the second party's MDE, wherein binding in the MDE of the components of the second object model and the set of metaprograms defines second software system code executable on a computer system having the architecture but not executable on a computer system having another architecture, the second software system code effecting a second user application when executed on the second computer system different from that which the first software system code effects when executed on the first computer system.

39. The method claimed in claim 38, wherein the object modeling computer language is the Unified Modeling Language (UML).

40. The method claimed in claim 38, wherein:

the step of the first party writing a first object model comprises a business analyst writing the first object model; and

the step of the second party writing a second object model comprises a business analyst writing the second object model.